Name: _____ CSCI 237

Practice Problems: x86_64 ASM

1.) Conditionals: Consider the following C function returns the larger of its two char arguments:

```
char return_larger(char x, char y)
{
   if (x > y)
      return x;
   else
      return y;
}
```

Using the `-Og` flag, the compiler generates the following assembly code:

```
    movl   %esi, %eax   # ret = __y__
    cmpb   %sil, %dil   # calc x - y, set cond codes
    jle    .L2          # jump if _x <= y_
    movl   %edi, %eax   # ret = __ x __
.L2:
    ret
```

a) Fill in the blanks using variables x, y, and appropriate logical operators to complete the comments associated with each line.

b) In the table below value, fill in the value of each condition code after executing the operation `cmpb %sil, %dil` for the given values of `x` and `y`:

x	***y***	***ZF***	***SF***	***OF***
2	*3*	*0*	*1*	*0*
-115	*100*	*0*	*0*	*1*
7	*7*	*1*	*0*	*0*
3	*2*	*0*	*0*	*0*
115	*-100*	*0*	*1*	*1*

c) What expression involving `ZF`, `SF`, and `OF` evaluates to `1` when the jump is taken and `0` otherwise? In other words, under what conditions does `jle` trigger a jump to the target?

`(SF^OF)|ZF.`

If subtracting caused a sign flip OR overflow but not both, then `x` is less than `y`. If the zero flag is set, then `x` and `y` are equal. So as long as either of those cases are true, then we want to jump.

Notes: using the standard C calling conventions, the first function argument is always stored in register `%rdi`, the second function argument in register `%rsi`, and the return value in register `%rax`.

The function `return_larger2` implements the same logic as `return_larger`, but it uses a ternary operator.

```
char return_larger2(char x, char y)
{
   return x > y ? x : y;
}
```

Compiling this function using the -Og flag produces the following (incomplete) assembly:

```
        cmpb    %dil, %sil
        movl    %edi, %eax
        cmovge  %esi, %eax
        ret
```

d) Complete the assembly by filling in the missing instruction, and explain what the completed function is doing in words.

It first compares the values of `x` and `y`. Then it "guesses" that `x` is larger by storing `x` in `%rax` (the return value). If the guess was wrong (`y >= x`), then it updates the return value to be `y`.

Note that `cmovge` and `cmovg` both yield the same result, but this version was chosen by the compiler.

2.) ASM to C: We have provided assembly code (left) that corresponds to the partially complete C function `foo` (right). Please fill in the missing lines of the C code.

```
    cmpq    %rsi, %rdi
    je    .L4
    movq    %rdi, %rax
    andq    %rsi, %rax
    ret
.L4:
    movq    %rdi, %rax
    imulq   %rsi, %rax
    ret
```

```
long foo (long a, long b)
{
  long result;
    if (a == b) {
    result = a * b;
  } else {
    result = a & b;
  }
  return result;
```

-OR-

```
    cmpq    %rsi, %rdi
    je    .L4
    movq    %rdi, %rax
    andq    %rsi, %rax
    ret
.L4:
    movq    %rdi, %rax
    imulq   %rsi, %rax
    ret
```

```
long foo (long a, long b)
{
  long result;
    if (a != b) {
    result = a & b;
  } else {
    result = a * b;
  }
  return result;
```

3.) **Loops:** Consider the general form of the C code on the left. Compiling it with command-line option -Og produces the assembly code on the right. Please fill in the missing lines of the C code. You may wish to use the box at the bottom to convert `bar` to an intermediate version (`bar_goto`) that uses `goto` statements.

```
long bar(long a, long b) {
  long result = __1__;

  while (___b > 0_____) {

    result = result * a_;

    b--_____________;
  }
  return result;
}
```

```
          movl    $1, %eax
          jmp     .L2
.L3:
          imulq   %rdi, %rax
          subq    $1, %rsi
.L2:
          testq   %rsi, %rsi
          jg      .L3
          ret
```

```
long bar_goto(long a, long b) {
  long result = ____1____;

  goto test____;

loop:

    result = result * a_;

    b--_____________;
test:

  if (__b > 0_____________)

        goto _loop_____;

  return result;
}
```

What does the function `bar` do, in words?

It computes a^b.

The following assembly code performs the same task, but it is derived from C code that uses different control structures. Finish the implementation C of `bar2`. You may wish to convert to a `goto` version as an intermediate step.

```
        movl    $0, %eax
        movl    $1, %edx
        jmp     .L2
.L3:
        imulq   %rdi, %rdx
        addl    $1, %eax
.L2:
        movslq  %eax, %rcx
        cmpq    %rsi, %rcx
        jl      .L3
        movq    %rdx, %rax
        ret
```

```
long bar2 (long a, long b) {
  long result = __1___;

  for(int i = 0; i < b; i++) {

     ____result *= a;_______

  }___________________

  return result;
}
```

Goto version:

```
long bar2_goto(long a, long b) {
  long result = 1;
  int i = 0;                    # init expression
  goto test;
loop:
    result = result * a;        # body statement
    i = i + 1;                  # update statement
    t = i < b;                  # test expression
    if (t)
        goto loop;
  return result;
}
```

Instead of a `while` loop, `bar2` uses a `for` loop to multiply an accumulator value by the value of `a` a total of `b` times.